XML/XMI Pros and Cons

Frank Farance
Farance/Edutool
1999-04-20
+1 212 486 4700
frank@farance.com
http://www.farance.com
http://edutool.com

Overview

- Progress since last meeting
- XMI (XMI Metadata Interchange), data models interchange, data interchange
- XML progress, data interchange work
- Conclusions
- Future work

Progress Since Last Meeting

- Contacted Sridhar Iyengar (Unisys), XMI co-author; discussed interoperability issues
- Standards participation: SC25, SC32, L8, IEEE, ISO
- Discuss XMI with other OMG people at 1999-03 meeting
- Conclusions: XMI supports data model interchange, not data interchange

XMI: XML Metadata Interchange

- XML Metadata ***Interchange***:
  - Exchange of information
- XML ***Metadata*** Interchange:
  - What type of information? Metadata ... descriptions of data models (typing, etc.)
- ***XML*** Metadata Interchange:
  - Using XML as a coding mechanism

Exchanging Data Models

- Simply put:
  - Structures (records)
  - Relationships
  - Data typing
  - Attribute
  - Other data model information

How Does Data Model Interchange Help?

- Examples:
  - Two CASE tools could exchange a data model, i.e., the same data model could be shared across various CASE tool vendors
  - Two repositories of *similar information*, but *implemented differently*, could share a common data model
  - Common data models can be described

Data Interchange

- Exchanges data, not typing (descriptive information)
- Exchanges “instances”, not “classes”
- Data interchange (usually) incorporates coding, encoding, and transfer methods

Data Model Interchange vs. Data Interchange

- Analogy to “portable code” in programming languages:
  - Data model interchange ==> common behavior *within in the context of a binding*
  - Data interchange ==> common behavior *across all bindings*

Example of Interchange

```
struct
{
    int a;
    float b;
    complex c;
} x;
read_x()
{
    read(fd,&x,sizeof x);
}
```

- Common data model? Yes
- Each implementation has same behavior? Yes
- Are implementations compatible? No
- ***Bindings*** are different

XML Progress

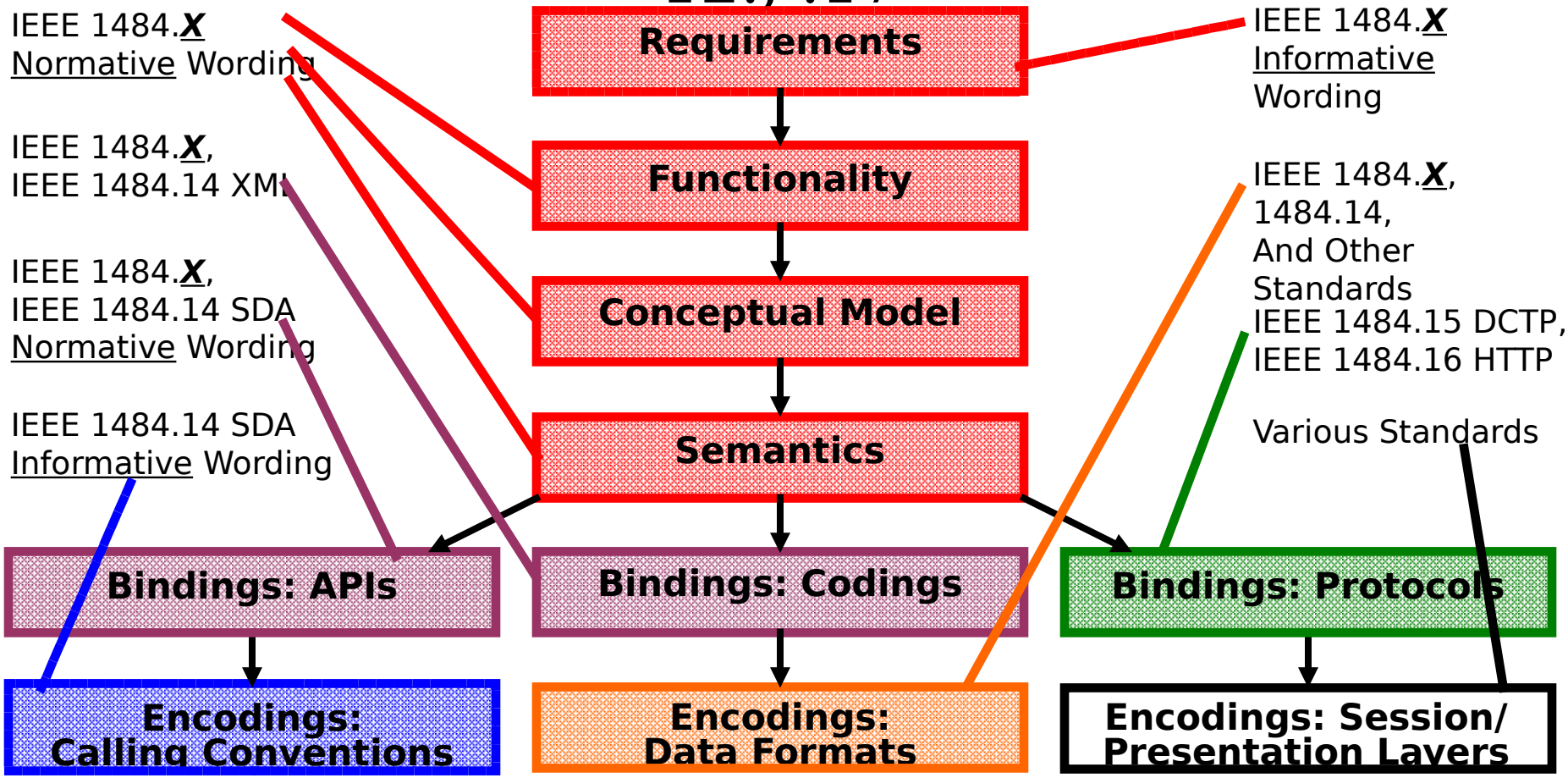
- Still problems with XML namespace:
  - Scoping issues not resolved
- Problems with global unique IDs
  - XML scoping
  - General scoping problem
  - Significant performance issues if GUIDs used
- Typing and labeling ambiguities

XML Progress

- Workarounds:
  - Don’t use GUID … use hierarchical names
  - Use “LABEL” attribute for “typed” tags
  - Use “TYPE” attribute for “identifier” tags
- Navigation issues are yet to be resolved
  - X pointers?

IEEE 1484.14: API and XML Bindings

IEEE 1484.***X*** is: .2, .4, .6, .7, .8, .10., 11., 12., .17



IEEE 1484.14: APIs and XML Bindings

- Define APIs: C, C++, Java, Javascript, Visual Basic, Perl, Tcl
- Contributions from several organization
- Collaboration with many standards organizations
- Related protocol work:
  - IEEE 1484.15 DCTP (data/control transfer protocol)
  - IEEE 1484.16 HTTP bindings

IEEE 1484.14: Session API Examples

- **`NewSession()`**: Opens a new session to repository
- **`DestroySession()`**: Closes repository.
- **`OpenView()`**: Starts data access to a portion of the repository.
- **`CloseView()`**: Ends data access.
- **`SetParam()`**, **`QueryParam()`**: Changes or retrieves session parameters.

IEEE 1484.14: Handlers API Examples

- **`SetHandler()`**, **`QueryHandler()`**: Set up and inquire about event handlers
- **`RequestSecurity()`**, **`RespondSecurity()`**: Allow “server-side” authentication requests
- **`RequestNomad()`**, **`RespondNomad()`**: Can setup nomadic (reconnectable) connections
- **`RequestEvent()`**, **`RespondEvent()`**: Used for handling most exceptions and errors

IEEE 1484.14: Data Perspectives API Examples

- **`SetNaming()`**, **`QueryNaming()`**: Set or inquire naming convention
- **`SetCoding()`**, **`QueryCoding()`**: Sets data coding, i.e., organization of information
- **`SetView()`**, **`QueryView()`**: Set or inquire view

IEEE 1484.14: Data Transfer/Walk API Examples

- **GetValue()**, **PutValue()**
- **GetDeepValue()**, **PutDeepValue()**
- **Copy()**, **DeepCopy()**
- **Move()**, **DeepMove()**
- **WalkGetObject()**, **WalkPutObject()**
- **List()**, **DeepList()**

Structured Data What Is It? [1/2]

- Known typing
- Known navigation
- Example:

```
record x
{
    integer p;
    string q;
    real r;
} y;
```

Structured Data What Is It? [2/2]

- Typing is known:

```
y ==> record x
p ==> integer
q ==> string
r ==> real
```

- Navigation known:

```
y.p    y.q    y.r
```

Unstructured Data What Is It? [1/3]

- Typing might not be known
- Navigation might not be known
- Example:

```
{ 17, “SMITH”, 0x80000000, 0x00000016,
6, 85, 77, 73, 86, 72, 0};
```

- Bytes, characters, strings, reals, integers?
- Internal structure?

Unstructured Data What Is It? [2/3]

- How do we know structure (type) is really:

```
record r
{
integer a;
character b[6];
real c;
string pointer d; // pointer to string
integer e; // length of string
} s;
```

Unstructured Data What Is It? [3/3]

- How do we know we can navigate data as:
  `s.d[0],s.d[1],s.d[2],...,s.d[s.e-1]`
- Unstructured data is very difficult to interpret in the general case

Semi-Structured Data What Is It? [1/4]

- Typing is known, OR
- Navigation is known
- Sometimes, but not always both
- Example 1:

```
<record>
<phone label="home">12125551212</phone>
<phone label="work">12125551000</phone>
</record>
```

Semi-Structured Data What Is It? [2/4]

- Type is known (**phone**)
- Navigation is unknown:
  - Will "**home**" always be first
  - How do I reference "**home**"?

```
record.home
record[0]
```

Semi-Structured Data What Is It? [3/4]

- Example 2:

```
<record>
<homephone>12125551212</homephone>
<workphone>12125551000</workphone>
<emergencyphone>12125559999</emergencyphone>
</record>
```

- Assume "`emergencyphone`" is an extension

Semi-Structured Data What Is It? [4/4]

- Navigation is known: **`homephone`**, **`workphone`**, **`emergencyphone`**
- Typing is unknown: what is the type of "**`emergencyphone`**"?

Semi-Structured Data Summary

- Data interoperability issues:
  - Need to define conceptual model of navigation and typing
  - XML conventions to support automated processing
  - May be able to interpret pieces of semi-structured data
  - Need to define conceptual model of interpretation

Conclusions

- Still need refinement on conceptual model
- Primary needs for data interoperability:
  - Required, optional, extensions
  - Subsets
  - Supersets
  - Revisions, multiple versions - compatibility
  - Typing
  - Navigation

Future Work

- COE meetings … demos/progress reports
- ANSI IISP Metadata and Data Warehousing Meeting: 1999-05-07
- SC32/WG2 1999-05 Meeting
- IEEE 1484.14 1999-05, 1999-09 Meetings
- NCITS/L8, multiple meetings this year